Scaling Kubernetes Networking Beyond 100,000 Endpoints

Rob Scott, Google
Minhan Xia, Google

Outline

- Problem Statement
- EndpointSlice API intro
- How it works
- Profiling Kubernetes
- Performance at 100k
- What’s next

Problem Statement

Service networking bottlenecks in Kubernetes

Current pain points

1. Limit for # of endpoints in a service
2. Performance degradation in large clusters

Existing Endpoints API


```
type Endpoints struct {
        metav1.TypeMeta `json:",inline"`
        // Standard object's metadata.
        // More info: https://git.k8s.io/community/contributors/devel/api-conventions.md#metadata
        // +optional
        metav1.ObjectMeta `json:"metadata,omitempty" protobuf:"bytes,1,opt,name=metadata"`

        // The set of all endpoints is the union of all subsets. Addresses are placed into
        // subsets according to the IPs they share. A single address with multiple ports,
        // some of which are ready and some of which are not (because they come from
        // different containers) will result in the address being displayed in different
        // subsets for the different ports. No address will appear in both Addresses and
        // NotReadyAddresses in the same subset.
        // Sets of addresses and ports that comprise a service.
        // +optional
        Subsets []EndpointSubset `json:"subsets,omitempty" protobuf:"bytes,2,rep,name=subsets"`
}
```

Endpoints per service limit

# of backend pods: **P**

Size of Endpoints object: **O(P)**

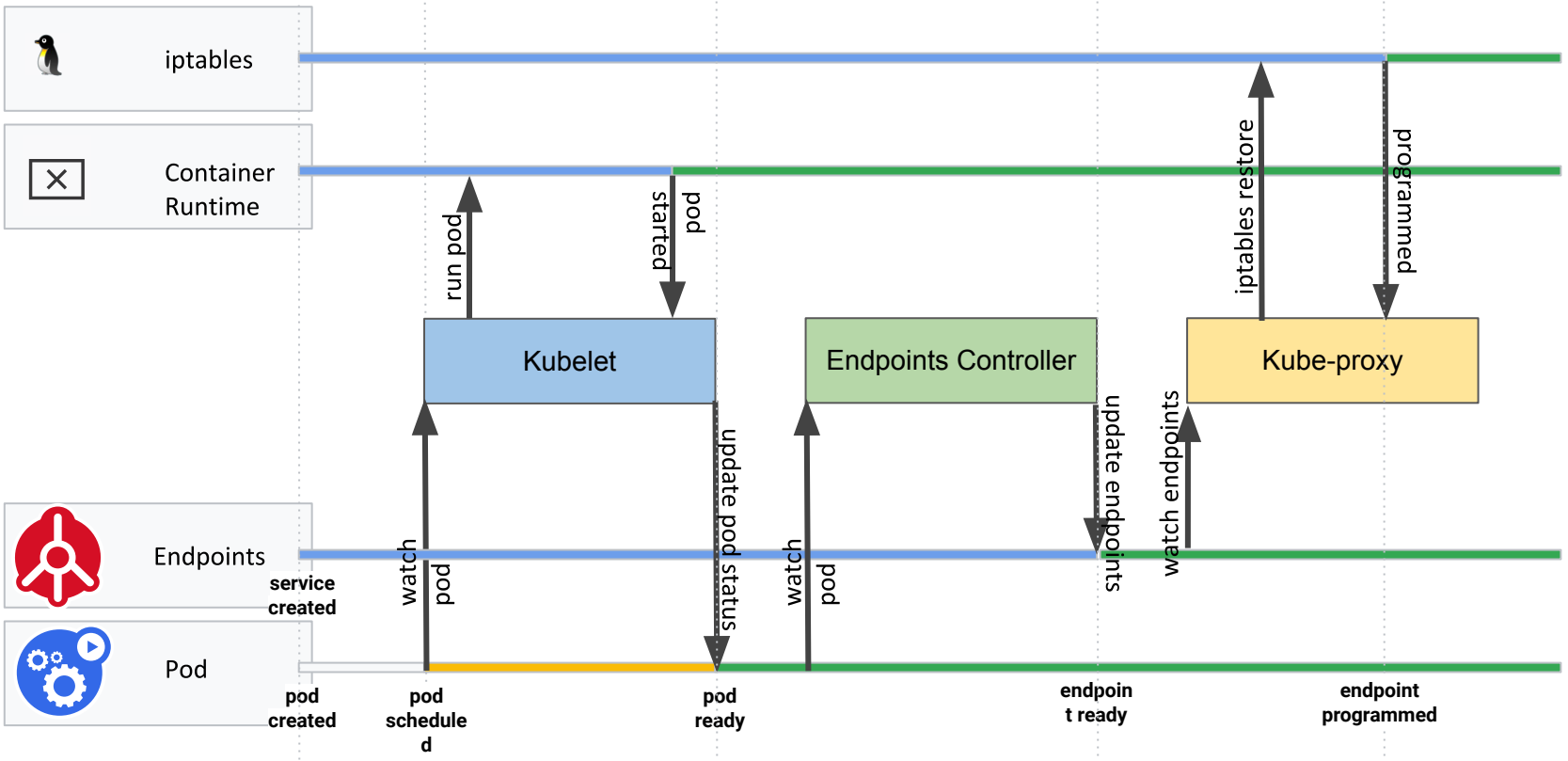
Max size of etcd object

1.5 MB ≈ O(5000) endpoints

Max size of etcd object

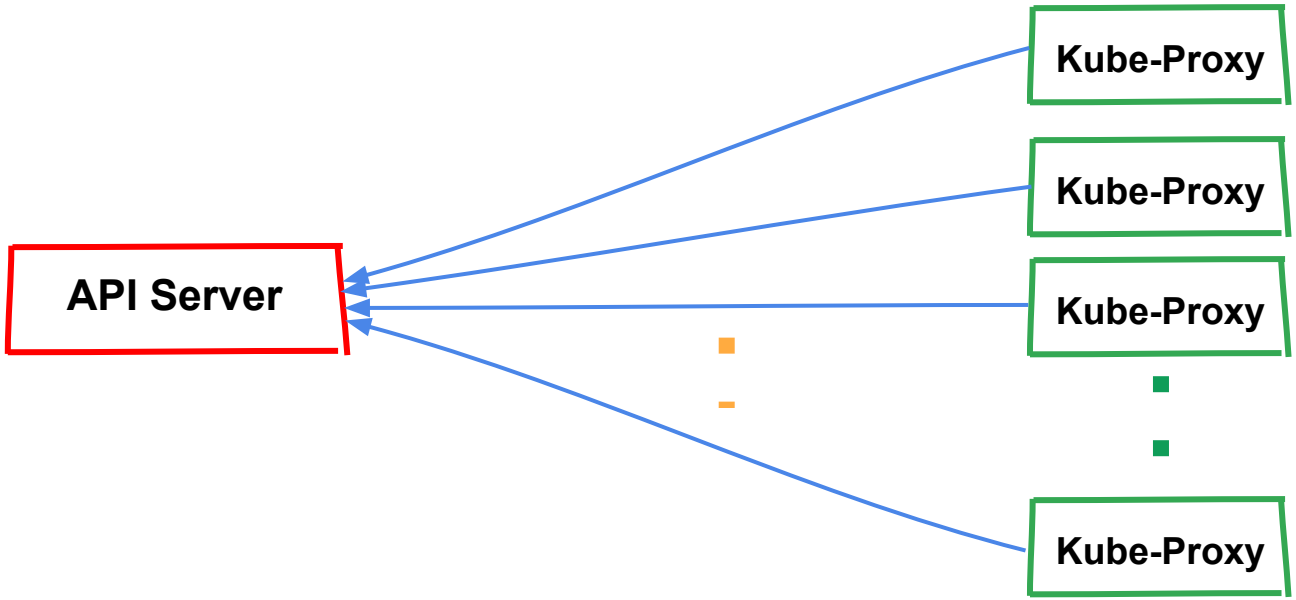
if endpoints object > 1.5 MB:

Service Control Flow



Performance Degradation

GET /api/v1/endpoints?watch=true&...



Performance Degradation

# of nodes: **N**

# of watchers: **N**

# of object copies per update: **N**

Performance Degradation

# of backend pods: **P**

Size of Endpoints object: **O(P)**

watchers: **N**

total bytes transmitted per update: **O(NP)**

Estimation

# of nodes: **5000**

Size of Endpoints object: **1 MB**

total bytes transmitted per update:

5000 X 1 MB = 5GB

DVD?

Estimation

total bytes transmitted per update: **5GB**

rolling update?

~5000 X 5 GB = 25TB !

User Expectations

- 10k+ endpoints/service
- Large Cluster
- High churn within a service

Just Works!

KubeCon | CloudNativeCon

Europe 2020

Virtual

EndpointSlice API intro

Scalable and extensible

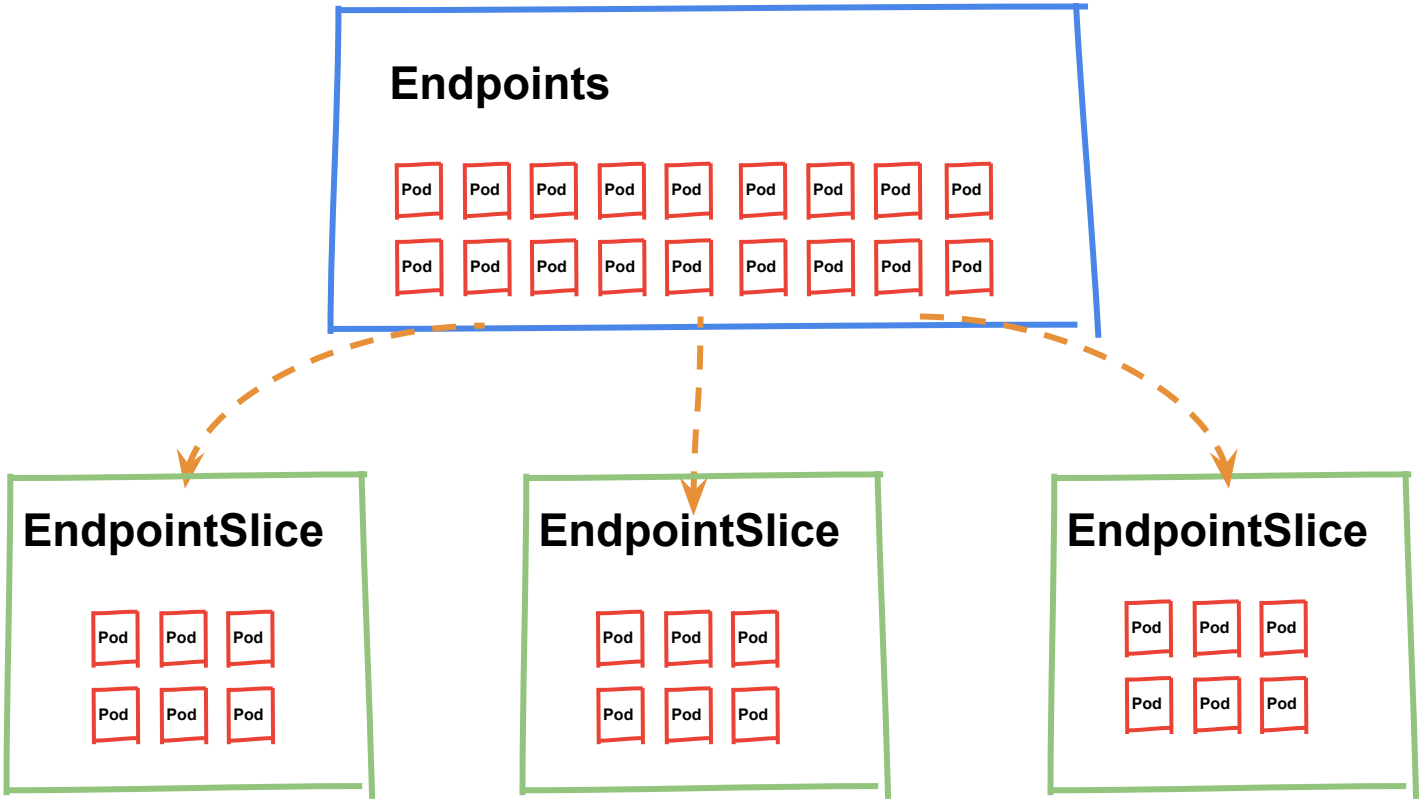
Goals

- Support **tens of thousands** of backend endpoints in a cluster with **thousands** of nodes

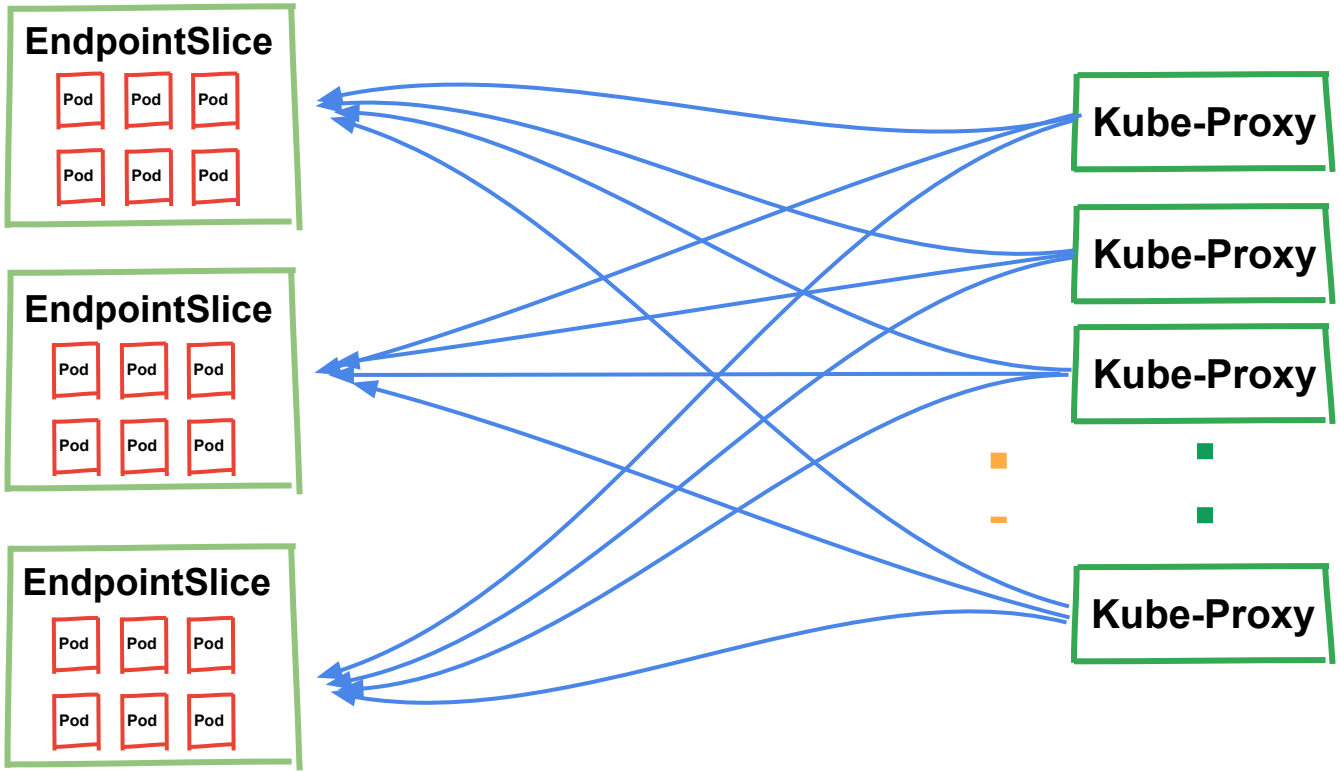
- Enable future extensions:
  - Dualstack
  - App Protocol
  - FQDN
  - Topological Aware Service
  - Dynamic endpoints subsetting
  - Multi-cluster Service discovery

  ...

High-level idea

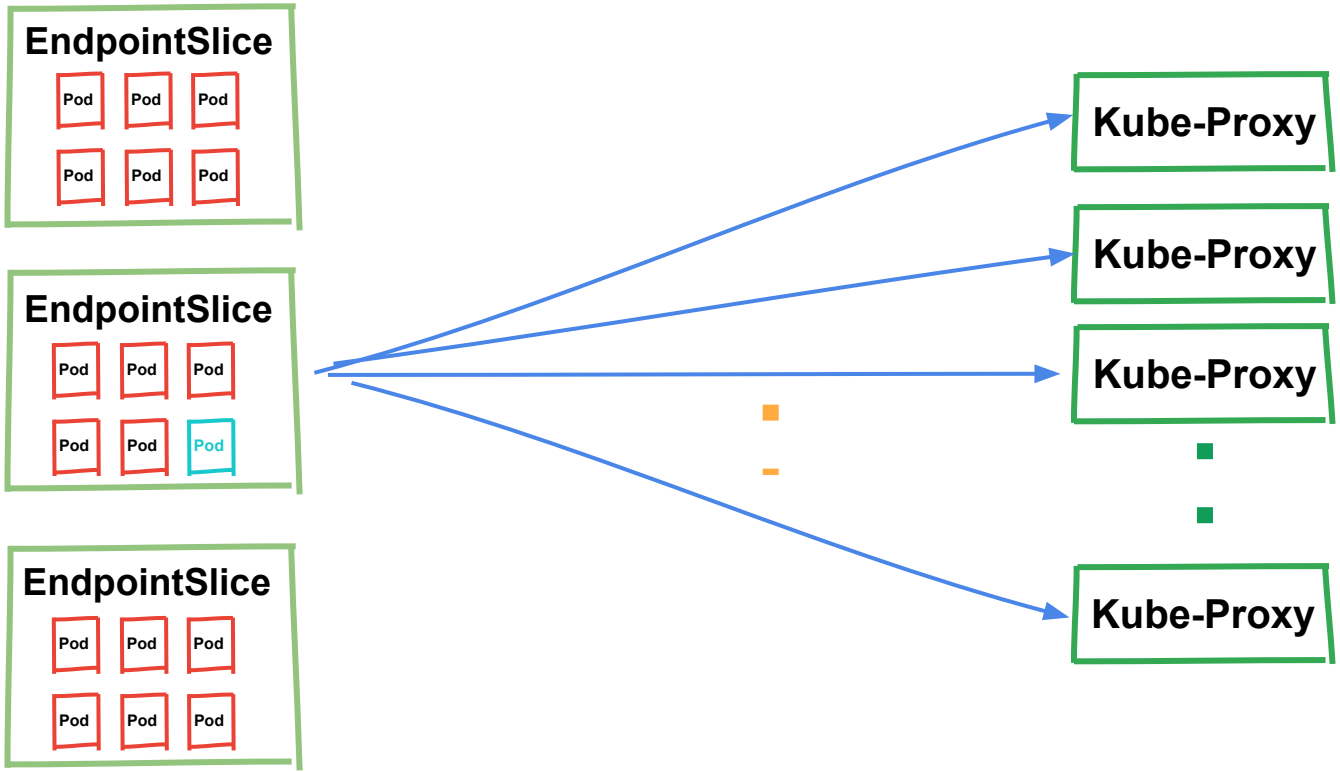


Endpoint Update



Endpoint Update

Endpoint Update



KubeCon | CloudNativeCon

Europe 2020

Virtual

How it works

The key components that make EndpointSlices work

Key Components

- **EndpointSlice controller:** Watches Services and Pods and creates or updates EndpointSlices
- **EndpointSliceMirroring controller:** Watches custom Endpoints and mirrors them to EndpointSlices
- **Kube-Proxy:** Watches Services and EndpointSlices and updates iptables or IPVS proxy rules

EndpointSlice Controller Goals

- Reduce EndpointSlice churn
  - Every node will watch EndpointSlices - updates are expensive
- Limit RPS to API Server
  - EndpointSlices that are too small will result in too many resources being updated
  - EndpointSlices that are too big will result in a DVD's worth of data getting sent across the cluster for even tiny changes

Example

New endpoints to add: 50

epslice-1
70 endpoints

epslice-2
80 endpoints

Example

Although they could all fit if we updated epslice-1 and epslice-2, we prefer a single create over multiple updates.

epslice-1
70 endpoints

epslice-2
80 endpoints

epslice-3
50 endpoints

KubeCon | CloudNativeCon

Europe 2020

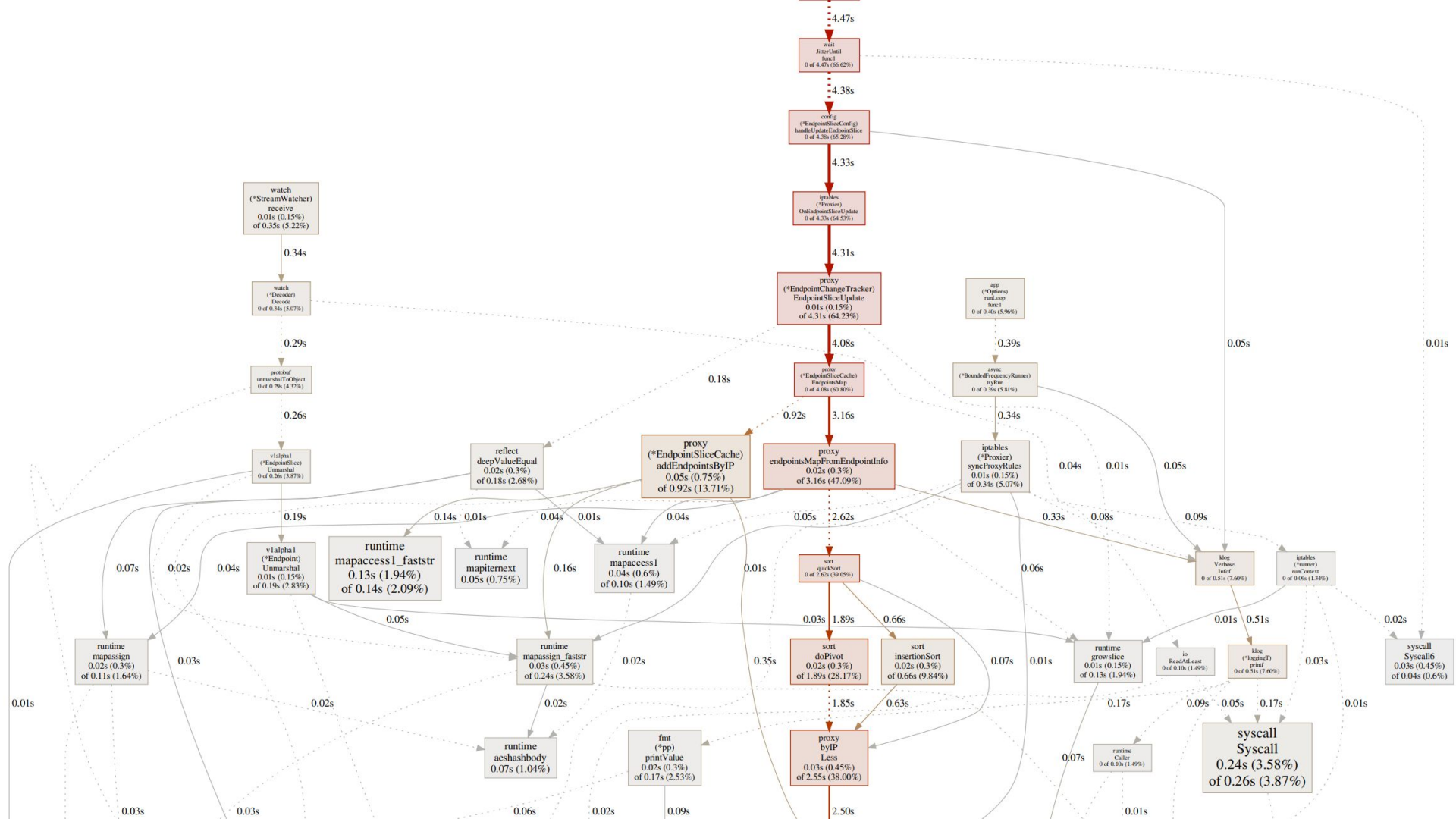
Virtual

Profiling Kubernetes

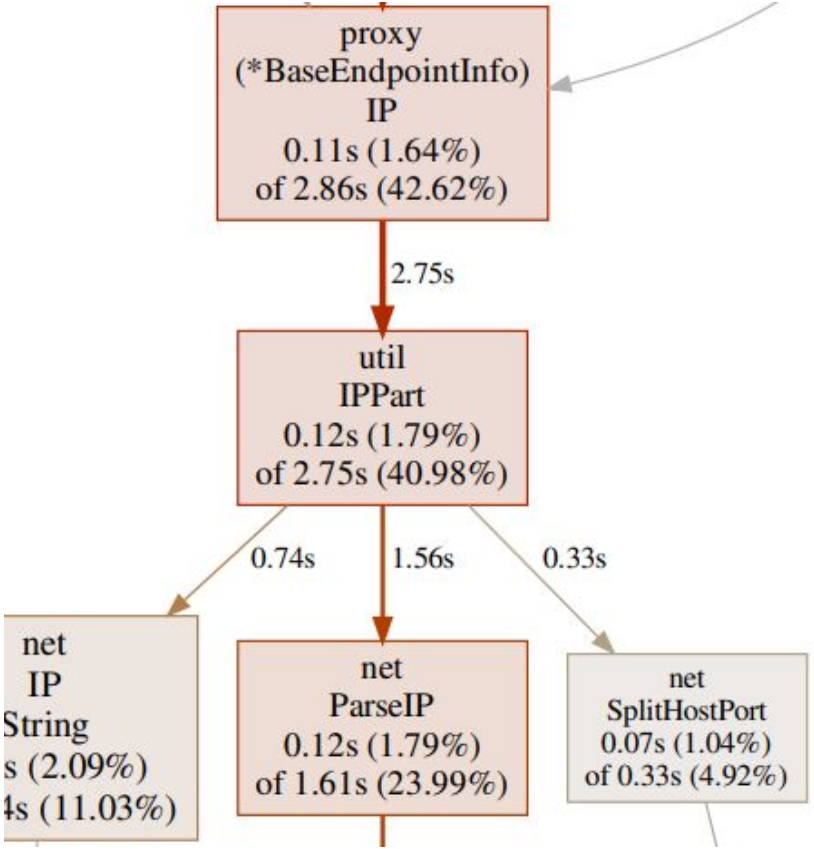
Finding bottlenecks in the codebase

The Problem

- kube-proxy was slower when using EndpointSlices than Endpoints
- The implementation was more complex
- EndpointSlice implementations needed to be faster
- Profiling kube-proxy with pprof was very helpful


4.47s
wait
JitterUntil
func1
0 of 4.47s (66.62%)
4.38s
config
(*EndpointSliceConfig)
handleUpdateEndpointSlice
0 of 4.38s (65.28%)
4.33s
iptables
(*Proxier)
OnEndpointSliceUpdate
0 of 4.33s (64.53%)
4.31s
proxy
(*EndpointChangeTracker)
EndpointSliceUpdate
0.01s (0.15%)
of 4.31s (64.23%)
watch
(*StreamWatcher)
receive
0.01s (0.15%)
of 0.35s (5.22%)
0.34s
watch
(*Decoder)
Decode
0 of 0.34s (5.07%)
0.29s
protobuf
unmarshalToObject
0 of 0.29s (4.32%)
0.26s
v1alpha1
(*EndpointSlice)
Unmarshal
0 of 0.26s (3.87%)
0.19s
v1alpha1
(*Endpoint)
Unmarshal
0.01s (0.15%)
of 0.19s (2.83%)
reflect
deepValueEqual
0.02s (0.3%)
of 0.18s (2.68%)
0.18s
proxy
(*EndpointSliceCache)
addEndpointsByIP
0.05s (0.75%)
of 0.92s (13.71%)
0.92s
proxy
(*EndpointSliceCache)
EndpointsMap
0 of 4.08s (60.80%)
4.08s
proxy
endpointsMapFromEndpointInfo
0.02s (0.3%)
of 3.16s (47.09%)
3.16s
app
(*Options)
runLoop
func1
0 of 0.40s (5.96%)
0.39s
async
(*BoundedFrequencyRunner)
tryRun
0 of 0.39s (5.81%)
0.34s
iptables
(*Proxier)
syncProxyRules
0.01s (0.15%)
of 0.34s (5.07%)
0.14s
0.01s
0.04s
0.01s
0.04s
0.05s
2.62s
0.04s
0.01s
0.05s
0.33s
0.08s
0.09s
0.05s
0.01s
runtime
mapaccess1_faststr
0.13s (1.94%)
of 0.14s (2.09%)
runtime
mapiternext
0.05s (0.75%)
runtime
mapaccess1
0.04s (0.6%)
of 0.10s (1.49%)
sort
quickSort
0 of 2.62s (39.05%)
klog
Verbose
Infof
0 of 0.51s (7.60%)
iptables
(*runner)
runContext
0 of 0.09s (1.34%)
0.07s
0.02s
0.04s
0.16s
0.01s
0.06s
0.03s
1.89s
0.66s
0.01s
0.51s
0.02s
runtime
mapassign
0.02s (0.3%)
of 0.11s (1.64%)
runtime
mapassign_faststr
0.03s (0.45%)
of 0.24s (3.58%)
sort
doPivot
0.02s (0.3%)
of 1.89s (28.17%)
sort
insertionSort
0.02s (0.3%)
of 0.66s (9.84%)
runtime
growslice
0.01s (0.15%)
of 0.13s (1.94%)
io
ReadAtLeast
0 of 0.10s (1.49%)
klog
(*loggingT)
printf
0 of 0.51s (7.60%)
syscall
Syscall6
0.03s (0.45%)
of 0.04s (0.6%)
0.03s
0.02s
0.35s
0.07s
0.01s
0.03s
0.01s
0.02s
0.02s
1.85s
0.63s
0.17s
0.09s
0.05s
0.17s
0.01s
runtime
aeshashbody
0.07s (1.04%)
fmt
(*pp)
printValue
0.02s (0.3%)
of 0.17s (2.53%)
proxy
byIP
Less
0.03s (0.45%)
of 2.55s (38.00%)
runtime
Caller
0 of 0.10s (1.49%)
syscall
Syscall
0.24s (3.58%)
of 0.26s (3.87%)
0.07s
0.03s
0.03s
0.06s
0.02s
0.09s
2.50s
0.01s

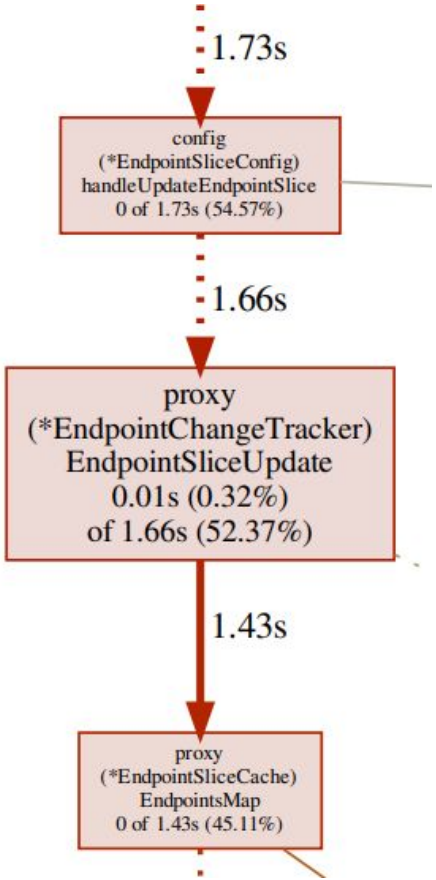
Results



endpoint.IP() was taking 43% of CPU time

- Method was used for sorting and diffing
- It didn't just return a string, it used netutil to parse an IP out of an IP:Port string

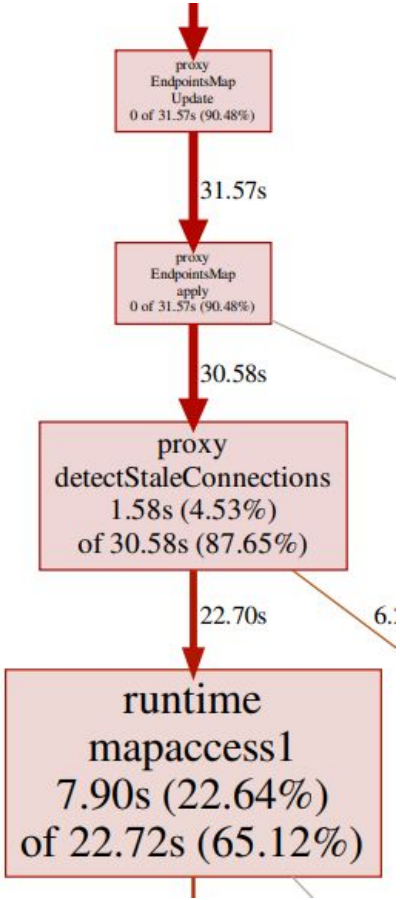
Results



handleUpdateEndpointSlice() was using 55% of CPU time

- We were calculating new EndpointsMap data structures every time an EndpointSlice was updated.
- This was only used when proxy rules were synced, much less frequently

Results



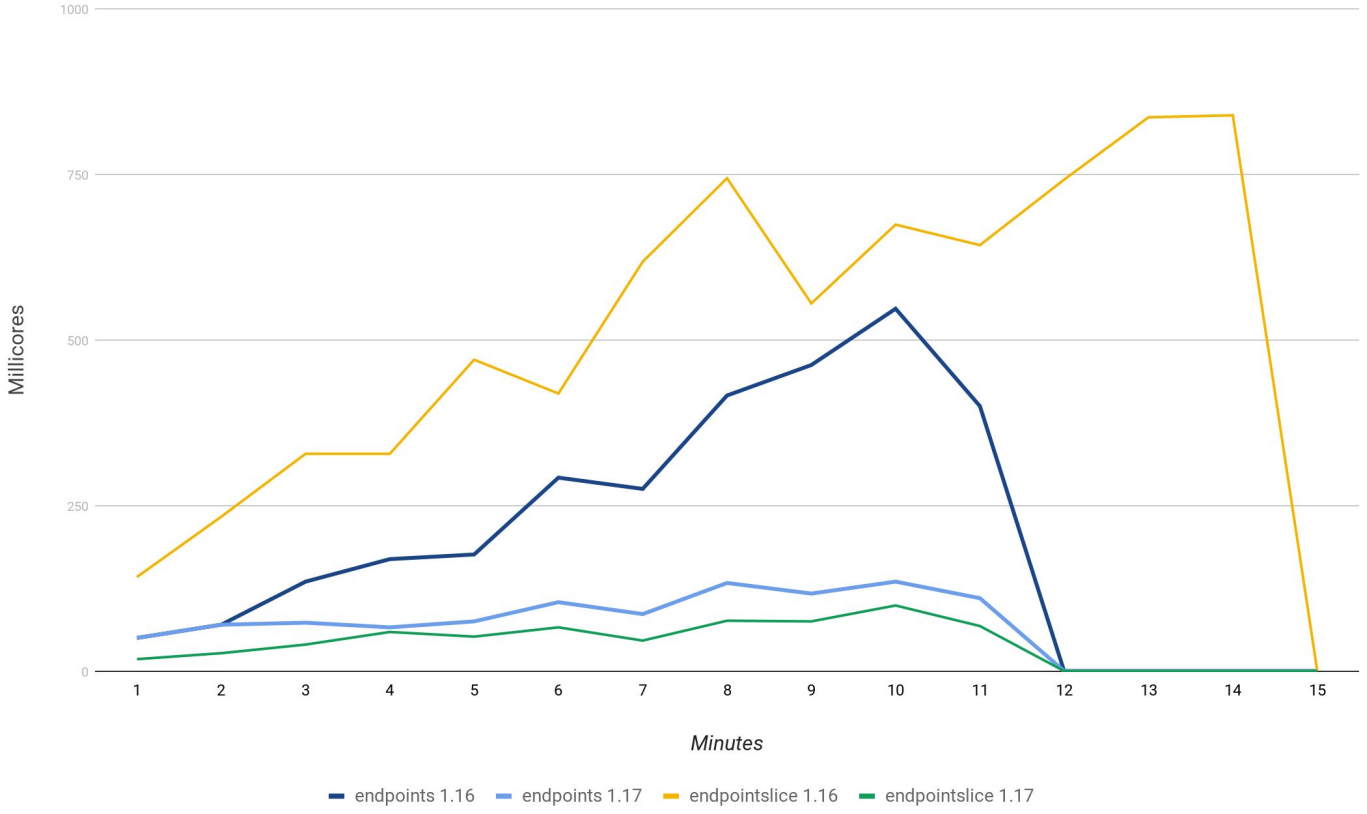
detectStaleConnections() was using 88% of CPU time

- Necessary to cleanup stale UDP connections
- Was running for all connections
- Restructured this so it only ran for UDP connections

Scaling to 10k endpoints

KubeCon CloudNativeCon Europe 2020 Virtual

Kube-Proxy CPU Usage



Scaling to 10k endpoints

Implementation	CPU time	% of baseline
Endpoints 1.16 (baseline)	116.7	100%
Endpoints 1.17	22.1	18.9%
EndpointSlice 1.16	312.5	260%
EndpointSlice 1.17	6.4	**5.4%**

KubeCon | CloudNativeCon

Europe 2020

Virtual

Performance at 100k

How big is too big?

Caveats

- These results are not scientific, we just wanted to know what would happen at this scale
- This is not an endorsement of running a Service with 100k endpoints in production
- We did nothing to tune the cluster for better performance at scale, it was running with all the standard kubetest settings
- With appropriate efforts to tune the master components, results would likely have been significantly better

Setup

- Used Kubetest with most defaults used for e2e test clusters
- 1.19 alpha prerelease `(v1.19.0-alpha.2.2611+a1a2f8c5f854e2)`

```
HEAPSTER_MACHINE_TYPE=n1-standard-32 \
kubetest --up \
--provider=gce \
--gcp-nodes=4001 \
--gcp-node-size=n1-standard-1 \
--gcp-zone=us-east1-a
```

Setup

- 4002 Nodes
- 1 Master
- 1 for Heapster
- 4000 Nodes for everything else

```
→ kubectl get nodes --no-headers | wc -l
4002
```

Setup

- 10 deployments * 10k pods each = 100k pods
- Divided into chunks so `kubectl get pods` won't timeout

```
→ kubectl get deploy
NAME          READY           UP-TO-DATE   AVAILABLE   AGE
scale-100ka   10000/10000     10000        10000       131m
scale-100kb   10000/10000     10000        10000       131m
scale-100kc   10000/10000     10000        10000       124m
scale-100kd   10000/10000     10000        10000       124m
scale-100ke   10000/10000     10000        10000       124m
scale-100kf   10000/10000     10000        10000       124m
scale-100kg   10000/10000     10000        10000       124m
scale-100kh   10000/10000     10000        10000       124m
scale-100ki   10000/10000     10000        10000       123m
scale-100kj   10000/10000     10000        10000       123m
```

Setup

- 1 NodePort Service targeting Pods for all deployments

```
→ kubectl get svc scale-100k
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
scale-100k    NodePort    10.0.187.81    <none>        80:32301/TCP   134m
```

Setup

- Endpoints hit default etcd object size limit at around 10k Pods
  - `Failed to update endpoint default/scale-100k: Request entity too large: limit is 3145728`

```
→ kubectl get endpoints scale-100k
NAME          ENDPOINTS                                                     AGE
scale-100k    10.64.1.12:9376,10.64.1.13:9376,10.64.1.14:9376 + 10053 more...   134m
```

```
Events:
  Type     Reason                 Age                 From                 Message
  ----     ------                 ----                ----                 -------
  Warning  FailedToUpdateEndpoint 59m (x16 over 63m)  endpoint-controller  Failed to update endpoint default/scale-100k: Request entity too large: limit is 3145728
```

Setup

- 1006 EndpointSlices to store 100,000 endpoints
- Controller minimizes updates and will always create a new EndpointSlice instead of updating multiple EndpointSlices

```
→ kubectl get endpointslice -l kubernetes.io/service-name=scale-100k --no-headers | wc -l
1006
```

```
→ kubectl get endpointslice -l kubernetes.io/service-name=scale-100k
NAME               ADDRESSTYPE   PORTS   ENDPOINTS                                      AGE
scale-100k-22r96   IPv4          9376    10.64.178.21,10.66.103.23,10.76.89.23 + 97 more...   114m
scale-100k-22x8z   IPv4          9376    10.74.160.2,10.75.118.2,10.76.5.2 + 97 more...      131m
scale-100k-24whl   IPv4          9376    10.68.93.4,10.71.45.4,10.71.85.4 + 97 more...       130m
scale-100k-25f62   IPv4          9376    10.74.43.16,10.76.94.16,10.68.85.16 + 97 more...    118m
scale-100k-25tkz   IPv4          9376    10.64.23.19,10.65.107.18,10.72.237.18 + 97 more...  117m
scale-100k-26jhs   IPv4          9376    10.69.213.4,10.67.249.4,10.73.62.4 + 97 more...     130m
scale-100k-29crw   IPv4          9376    10.67.186.19,10.71.145.19,10.78.245.18 + 97 more...  116m
scale-100k-2b26h   IPv4          9376    10.73.14.23,10.74.204.23,10.67.114.23 + 97 more...  113m
scale-100k-2bg68   IPv4          9376    10.73.32.7,10.69.145.7,10.66.70.7 + 97 more...      124m
scale-100k-2cn7h   IPv4          9376    10.77.12.24,10.70.228.24,10.64.209.24 + 79 more...  113m
```

So iptables?

- >400k total lines, >100k probabilities
- syncProxyRules: ~16s, iptables save and restore: ~9s

```
e2e-test-robertjscott-minion-heapster /home/robertjscott # iptables-save | wc -l
401964
e2e-test-robertjscott-minion-heapster /home/robertjscott # iptables-save | grep probability | wc -l
100450
```

```
Trace[1686045414]: [6.397507686s] [6.397507686s] END
I0722 00:37:47.799359       1 proxier.go:809] syncProxyRules took 16.41452803s
I0722 00:37:48.365944       1 proxier.go:845] Syncing iptables rules
I0722 00:37:57.685306       1 trace.go:205] Trace[870364468]: "iptables save" (22-Jul-2020 00:37:00.284) (total time: 2401ms):
Trace[870364468]: [2.401212728s] [2.401212728s] END
I0722 00:38:04.408488       1 trace.go:205] Trace[437202518]: "iptables restore" (22-Jul-2020 00:37:00.085) (total time: 6323ms):
Trace[437202518]: [6.323155055s] [6.323155055s] END
I0722 00:38:04.408556       1 proxier.go:809] syncProxyRules took 16.609181768s
I0722 00:38:04.960873       1 proxier.go:845] Syncing iptables rules
I0722 00:38:13.939897       1 trace.go:205] Trace[654215419]: "iptables save" (22-Jul-2020 00:38:00.782) (total time: 2157ms):
Trace[654215419]: [2.157123083s] [2.157123083s] END
I0722 00:38:20.642612       1 trace.go:205] Trace[370910750]: "iptables restore" (22-Jul-2020 00:38:00.330) (total time: 6311ms):
Trace[370910750]: [6.311652576s] [6.311652576s] END
I0722 00:38:20.642657       1 proxier.go:809] syncProxyRules took 16.234085575s
I0722 01:01:07.623125       1 trace.go:205] Trace[1698162333]: "iptables Monitor CANARY check" (22-Jul-2020 01:01:00.716) (total time: 2906ms):
Trace[1698162333]: [2.906939806s] [2.906939806s] END
I0722 01:38:20.649680       1 proxier.go:845] Syncing iptables rules
I0722 01:38:29.961032       1 trace.go:205] Trace[685925543]: "iptables save" (22-Jul-2020 01:38:00.727) (total time: 2233ms):
Trace[685925543]: [2.233227673s] [2.233227673s] END
I0722 01:38:36.515454       1 trace.go:205] Trace[1740200389]: "iptables restore" (22-Jul-2020 01:38:00.237) (total time: 6278ms):
Trace[1740200389]: [6.278283666s] [6.278283666s] END
I0722 01:38:36.515550       1 proxier.go:809] syncProxyRules took 15.872611711s
```

Results: iptables

KubeCon | CloudNativeCon Europe 2020 Virtual

```
Summary:
  Total:        201.0422 secs
  Slowest:      1.8651 secs
  Fastest:      0.0003 secs
  Average:      0.8831 secs
  Requests/sec: 248.7041

  Total data:   1390022 bytes
  Size/request: 27 bytes

Response time histogram:
  0.000 [1]     |
  0.187 [1185]  |
  0.373 [7813]  |
  0.560 [7336]  |
  0.746 [5708]  |
  0.933 [9959]  |
  1.119 [3867]  |
  1.306 [2905]  |
  1.492 [1969]  |
  1.679 [2797]  |
  1.865 [6460]  |


Latency distribution:
  10% in 0.3383 secs
  25% in 0.4644 secs
  50% in 0.7948 secs
  75% in 1.2440 secs
  90% in 1.7221 secs
  95% in 1.7406 secs
  99% in 1.7818 secs
```

```
[ root@63ee2c82cfc0:/ ]$ cat hey.txt | grep scale-100k | wc -l
12700
```

- 50k requests with `freehan/hey`
- Avg 0.8831s response time
- Requests went to 12,700 pods

```
Details (average, fastest, slowest):
  DNS+dialup:   0.1177 secs, 0.0003 secs, 1.8651 secs
  DNS-lookup:   0.0000 secs, 0.0000 secs, 0.0000 secs
  req write:    0.0001 secs, 0.0000 secs, 0.0404 secs
  resp wait:    0.0016 secs, 0.0001 secs, 0.4387 secs
  resp read:    0.1845 secs, 0.0000 secs, 1.5432 secs

Status code distribution:
  [200] 50000 responses

Response distribution:
  [scale-100ka-6cfcd99778-22qxb]        1 responses
  [scale-100ka-6cfcd99778-24cdq]        4 responses
  [scale-100ka-6cfcd99778-24lk6]        1 responses
  [scale-100ka-6cfcd99778-24lp6]        2 responses
  [scale-100ka-6cfcd99778-24qsh]        1 responses
  [scale-100ka-6cfcd99778-25bws]        1 responses
  [scale-100ka-6cfcd99778-26cm5]        33 responses
  [scale-100ka-6cfcd99778-26mlj]        4 responses
  [scale-100ka-6cfcd99778-274rm]        10 responses
  [scale-100ka-6cfcd99778-28mp5]        7 responses
  [scale-100ka-6cfcd99778-28sft]        32 responses
  [scale-100ka-6cfcd99778-28z6x]        1 responses
  [scale-100ka-6cfcd99778-29crj]        1 responses
  [scale-100ka-6cfcd99778-2brrn]        1 responses
  [scale-100ka-6cfcd99778-2chbj]        1 responses
```

Results: IPVS

```
Summary:
  Total:        201.3850 secs
  Slowest:      3.3511 secs
  Fastest:      0.0003 secs
  Average:      1.7456 secs
  Requests/sec: 248.2807

  Total data:   1390078 bytes
  Size/request: 27 bytes

Response time histogram:
  0.000 [1]     |
  0.335 [3410]  |
  0.670 [6042]  |
  1.006 [5509]  |
  1.341 [11348] |
  1.676 [1152]  |
  2.011 [1671]  |
  2.346 [1323]  |
  2.681 [4182]  |
  3.016 [4574]  |
  3.351 [10788] |

Latency distribution:
  10% in 0.4340 secs
  25% in 0.8871 secs
  50% in 1.2703 secs
  75% in 2.7876 secs
  90% in 3.1790 secs
  95% in 3.2277 secs
  99% in 3.2474 secs
```

```
[ root@63ee2c82cfc0:/ ]$ cat hey.txt | grep scale-100k | wc -l
17138
```

- 50k requests with `freehan/hey`
- Avg 1.7456s response time
- Requests went to 17,138 pods

```
Details (average, fastest, slowest):
  DNS+dialup:   0.3690 secs, 0.0003 secs, 3.3511 secs
  DNS-lookup:   0.0000 secs, 0.0000 secs, 0.0000 secs
  req write:    0.0002 secs, 0.0000 secs, 0.1272 secs
  resp wait:    0.0019 secs, 0.0001 secs, 0.4480 secs
  resp read:    0.3300 secs, 0.0000 secs, 3.0679 secs

Status code distribution:
  [200] 50000 responses

Response distribution:
  [scale-100ka-6cfcd99778-228jn]      1 responses
  [scale-100ka-6cfcd99778-22qxb]      4 responses
  [scale-100ka-6cfcd99778-22t78]      1 responses
  [scale-100ka-6cfcd99778-247vk]      1 responses
  [scale-100ka-6cfcd99778-24849]      3 responses
  [scale-100ka-6cfcd99778-24qsh]      3 responses
  [scale-100ka-6cfcd99778-24sjr]      1 responses
  [scale-100ka-6cfcd99778-25jvp]      1 responses
  [scale-100ka-6cfcd99778-267f9]      1 responses
  [scale-100ka-6cfcd99778-26mlj]      9 responses
  [scale-100ka-6cfcd99778-274j4]      8 responses
  [scale-100ka-6cfcd99778-275jf]      1 responses
  [scale-100ka-6cfcd99778-27bhc]      1 responses
  [scale-100ka-6cfcd99778-286jf]      2 responses
  [scale-100ka-6cfcd99778-28j6d]      1 responses
```

More than 100k

- It all continues to work with 120k Pods
- 1204 total EndpointSlices

```
λ robertjscott [src/k8s.io/hack] → kubectl get endpointslice -l kubernetes.io/service-name=scale-100k --no-headers | wc -l
1204
```

```
λ robertjscott [src/k8s.io/hack] → kubectl get deploy
NAME          READY          UP-TO-DATE   AVAILABLE   AGE
scale-100ka   10000/10000    10000        10000       5h5m
scale-100kb   10000/10000    10000        10000       5h5m
scale-100kc   10000/10000    10000        10000       4h58m
scale-100kd   10000/10000    10000        10000       4h58m
scale-100ke   10000/10000    10000        10000       4h58m
scale-100kf   10000/10000    10000        10000       4h58m
scale-100kg   10000/10000    10000        10000       4h58m
scale-100kh   10000/10000    10000        10000       4h58m
scale-100ki   10000/10000    10000        10000       4h57m
scale-100kj   10000/10000    10000        10000       4h57m
scale-100kl   10000/10000    10000        10000       16m
scale-100km   10000/10000    10000        10000       16m
```

Errors at Scale

API Server timeouts when setting up new watches

```
apiserver panic'd on GET
/apis/discovery.k8s.io/v1beta1/endpointslices?labelSelector=%21service.kubernetes.io%2Fheadless%2C%21service.kubernetes.io%2Fservice-proxy-name&resourceVersion=8653734
http2: panic serving 34.75.50.56:47304:killing connection/stream because serving request timed out and response had been started
```

EndpointSlice update fails - informer cache is out of date

```
"Event occurred" object="default/scale-100k" kind="Service" apiVersion="v1" type="Warning"
reason="FailedToUpdateEndpointSlices" message="Error updating Endpoint Slices for Service default/scale-100k:
[Error updating scale-100k-6kzx6 EndpointSlice for Service default/scale-100k: Operation cannot be fulfilled on
endpointslices.discovery.k8s.io \"scale-100k-6kzx6\":the object has been modified; please apply your changes
to the latest version and try again
```

Endpoints controller kept running into etcd object size limit - Endpoints will be truncated in future

```
Failed to update endpoint default/scale-100k: Request entity too large: limit is 3145728
```

Modes Compared

	ipvs		iptables	
	100k	**120k**	**100k**	**120k**
Avg response time (s)	1.7456	1.7468	0.8831	1.8891
Endpoint distribution	17,138	18,184	12,700	20,189
Avg update time (s)	~5	~5	~25	~29

These results are not scientific. They represent individual runs from a single node in a cluster with prerelease software. A variety of additional factors could affect these values.

What's Next?

The features EndpointSlices will enable

Goals for Kubernetes 1.20

- Automatic Topology Aware Routing Alpha
- EndpointSlice Subsetting for Kube-Proxy Alpha
- MultiCluster Services Alpha
- Significant dual stack updates
- EndpointSlice Windows Kube-Proxy Beta

Topology Aware Routing

- EndpointSlices store topology information (zone, region) for each endpoint
- Kube-Proxy can be updated to prefer endpoints that are in the same zone or region
- Potential for faster routing and significant cost savings

EndpointSlice Subsetting

- Controller can start to group EndpointSlices by unique topology keys such as zone and region
- Kube-Proxy can choose to select the subset of EndpointSlices that represent closest Endpoints
- Potential for huge performance improvements:
  - In a 3-zone cluster nearly 3x less endpoints for kube-proxy to watch and process
  - Significant decrease for API server load in large clusters

Conclusion

- Even if you don’t want to run a 100k endpoint Service, these performance improvements will be noticeable at all levels
- The upper limits of Service size are dramatically higher now
- EndpointSlices don't solve all the bottlenecks
- New features like topology aware routing and subsetting will result in significant scalability improvements


KubeCon
CloudNativeCon
Europe 2020
Virtual
HELM
KEEP CLOUD NATIVE
CONNECTED